Analysis of Dynamic Voltage/Frequency Scaling in Chip-Multiprocessors

Sebastian Herbert
Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
sherbert@ece.cmu.edu

Diana Marculescu
Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
dianam@ece.cmu.edu

ABSTRACT

Fine-grained dynamic voltage/frequency scaling (DVFS) demonstrates great promise for improving the energy-efficiency of chip-multiprocessors (CMPs), which have emerged as a popular way for designers to exploit growing transistor budgets. We examine the tradeoffs involved in the choice of both DVFS control scheme and method by which the processor is partitioned into voltage/frequency islands (VFIs). We simulate real multithreaded commercial and scientific workloads, demonstrating the large real-world potential of DVFS for CMPs. Contrary to the conventional wisdom, we find that the benefits of per-core DVFS are not necessarily large enough to overcome the complexity of having many independent VFIs per chip.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design studies

General Terms

Design, Performance

Keywords

Dynamic voltage/frequency scaling, chip-multiprocessor

1. INTRODUCTION

1.1 Overview and Related Work

Challenged with finding ways to use an ever-growing transistor budget, microarchitects have begun to find that the traditional methods of creating a more powerful out-of-order core have become less attractive. However, the chip-multiprocessor (CMP) offers an attractive solution. Multiple cores are replicated on a single die, resulting in little to no increase in local power density and potentially linear scaling of performance for multithreaded workloads.

CMPs are particularly well-suited to being implemented as systems composed of multiple independently-clocked voltage/frequency islands (VFIs). The division of a CMP into cores and shared cache provides a natural granularity for the VFI partitioning, such that the performance cost of using the VFI organization is negligible. However, significant improvements in energy-efficiency are enabled by fine-grained dynamic voltage/frequency scaling (DVFS).

There have been few studies on DVFS for CMPs, and almost all have assumed per-core DVFS [6][7]. However, commercial designs have traditionally used full-chip DVFS, only recently making the move to per-core frequency (but not voltage) scaling with Advanced Micro Devices' quad-core Opteron [5]. It is also no longer the case that only full-chip DVFS is available, which has been assumed elsewhere [8]. While using a small number of independently-clocked cores has been shown to be tractable by example, it is unclear that per-core DVFS will be complexity-effective as the number of cores per chip continues to scale upward.

1.2 Paper Contributions

Prior studies have been limited to using small benchmark sets such as SPEC2000 for their performance evaluations [6][7]. However, these represent a very limited usage scenario, so it is unclear how effective the proposed schemes would be in real-world deployments. We run real multithreaded commercial and scientific workloads on a full-system simulator, showing that the energy-efficiency gains for such workloads are significantly larger than those that have previously been demonstrated. Li and Martínez evaluated some parallel applications [8], but not commercial ones. Moreover, they performed optimization at the level of well-defined parallel regions, so their methodology is not extensible to many commercial application classes that lack these.

This paper makes several key contributions:

- We have created a significant amount of infrastructure to support our experiments, including an *integrated* performance/power/thermal simulator for both fully-synchronous and VFI CMPs capable of running real multithreaded operating systems and applications.

- We demonstrate the energy-efficiency gains that can be achieved on real workloads by adding fine-grained DVFS to a CMP, with the best design achieving an average energy/throughput^2 reduction of 38.2% relative

This research has been supported in part by Semiconductor Research Corporation contract No. 2005-HJ-1314.

ISLPED'07, August 27–29, 2007, Portland, Oregon, USA.

to the fully-synchronous baseline with no voltage or frequency scaling. By varying the DVFS control algorithm while keeping other parameters fixed, we provide a fair comparison of previously proposed schemes.

- We provide analyses of DVFS performance for several application classes (web server, online transaction processing, decision support systems, and scientific).
- We demonstrate that the benefits offered by doing DVFS at the core granularity rather than on clusters of cores are typically small and likely not worth the increased design complexity.

The remainder of this paper is organized as follows. Section 2 presents our hardware design, while Section 3 describes the infrastructure we use to simulate it. Sections 4 and 5 detail the DVFS algorithms and VFI configurations evaluated, respectively. Section 6 presents our experimental methodology and Section 7 our results. Section 8 concludes.

2. HARDWARE DESIGN

The starting point for our VFI-CMP design is a fully-synchronous CMP composed of 16 out-of-order cores. Its Piranha-style cache hierarchy [1] consists of private L1s and a logically shared L2 organized as 16 independent banks. We divide the processor into two, five, or 17 VFIs. One contains the L2 and memory interface, while the cores are split evenly among the remaining VFIs. As we are interested in isolating the effects of the partitioning of cores into VFIs, the memory domain operates at a fixed voltage and frequency.

Because the VFI interface logic is only required at the core-to-L2 interfaces, the VFI partitioning results in negligible performance loss due to the relative infrequency and long latency of L2 accesses. To perform interdomain communication, we use asynchronous, level-converting FIFO queues [4]. As long as the FIFO is neither full nor empty, the cell to be used for the next read differs from that to be used for the next write, allowing the two to proceed in parallel.

As in previous VFI studies [9], we assume that each clock domain uses a clock multiplier to create its local clock signal from that produced by a single, global PLL. Each domain also contains its own voltage regulator, allowing extremely fast transitions between voltage levels. Given the small gap between V_{dd} and V_{th} in modern technologies, our 45 nm design uses a relatively limited set of five evenly-spaced VF levels. V_{dd} can be reduced from the nominal value of 1.0 V to 0.6 V, with frequency decreasing from 4.0 GHz to 1.9 GHz. We assume that transitions can only be made between adjacent levels.

3. SIMULATION INFRASTRUCTURE

We use the Flexus CMPFlex.OoO simulator, which adds a detailed timing model to Virtutech Simics. Flexus is a full-system simulator which runs real workloads and operating systems and models all components of a computer system.

We added a hybrid instruction-/microarchitecture-level Wattch-like [2] dynamic power model to Flexus. We use the static power model proposed by Butts and Sohi [3], which uses a nominal leakage value along with estimates of the number of transistors (scaled by some design-dependent factors) in each microarchitectural structure. We model the dependence of subthreshold leakage on both temperature and threshold voltage, which is itself a function of temperature.

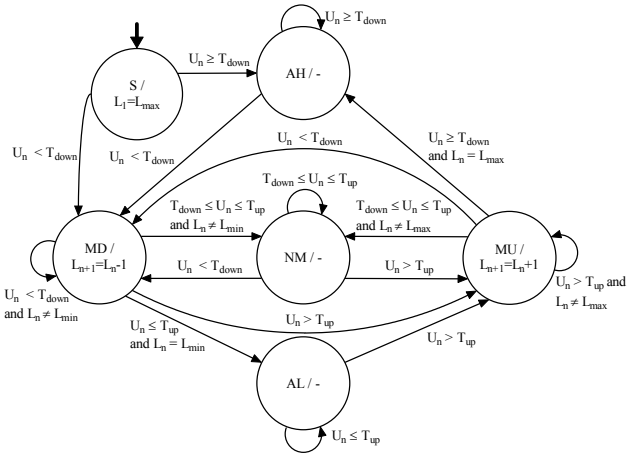


Figure 1: FSM for *Threshold* controller, also used for *PI*

We use the HotSpot thermal simulation package [11], with our floorplan consisting of a ring of 16 cores around central L2 cache banks. To account for the feedback loop between leakage power and temperature, we iterate simulations until power and temperature values converge, feeding the output steady-state temperatures of one run back in as the initial temperatures of the next.

We created a simulator for the VFI design outlined in Section 2. Each domain has a power model for its clock network that is based on its physical size and number of pipeline registers. We added a `FifoQueue` component to Flexus to model the latency penalty of crossing a VFI boundary.

4. DVFS ALGORITHMS

4.1 *Threshold*

We consider several DVFS control algorithms. *Threshold* is a simple threshold-based algorithm based on that proposed by Talpes and Marculescu [12]. A VFI's utilization over an interval, computed as the number of non-spin instructions retired divided by the number of retire slots, is compared to a pair of thresholds at the end of each interval. If it is higher (lower) than the up (down) threshold T_{up} (T_{down}), the VFI's voltage and frequency are increased (decreased). While a VFI's utilization remains in the target range $[T_{down}, T_{up}]$, its voltage and frequency are unchanged.

The FSM for the controller is shown in Figure 1; the states are `Start` (`S`), `At Highest` (`AH`), `At Lowest` (`AL`), `Move Down` (`MD`), `Move Up` (`MU`), and `No Move` (`NM`). U_n is the utilization and L_n is the VF level for interval n.

4.2 *PI*

PI is a control-theoretic technique using a proportional-integral controller, first proposed for single-core multiple-clock-domain processors by Wu *et al.* [15]. A distributed version for CMPs was proposed by Juang *et al.* [7], but their experimental context differs from ours. They investigated small benchmarks and kernels with lightweight threads on a multicore ARM derivative, while we examine real workloads with much larger threads on a high-performance microprocessor. Their distributed scheme requires the setting of per-thread load factors either by the compiler or through hand analysis. However, our workloads are extremely complex and their source code is not generally freely available.

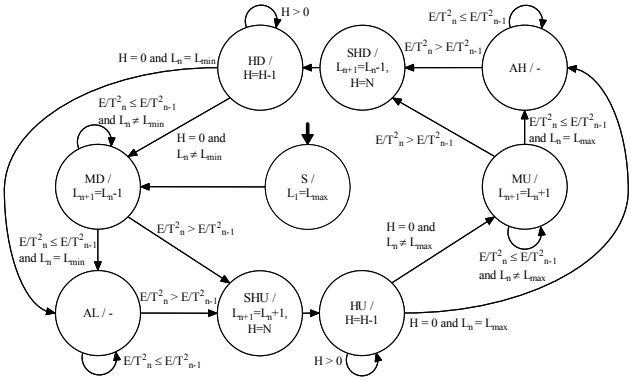


Figure 2: FSM for *Greedy* controller

Our threads also tend to interact less strongly than those of small, aggressively-multithreaded benchmarks and kernels. Thus, we use per-VFI PI controllers for the utilization, computed the same way as for *Threshold*.

The FSM for *PI* is the same as the one for *Threshold*, shown in Figure 1. U_n is replaced with f_{n+1}, the target frequency for the next interval computed by the PI controller. We want to raise (lower) the VF level if the next higher (lower) level's frequency is closer to f_{n+1} than the current one's. Thus, T_{up} is replaced with $\frac{f(L_n)+f(L_n+1)}{2}$ and T_{down} with $\frac{f(L_n)+f(L_n-1)}{2}$. The function $f(L)$ gives the frequency that results when running at level L (∞ for $L > L_{max}$ and $-\infty$ for $L < L_{min}$), so $\frac{f(L_n)+f(L_n+1)}{2}$ is the midpoint of the frequencies at the current level and the one above.

4.3 *Greedy*

Greedy is an adaptation of the *Greedy-search* method [9], which attempts to find the operating point with minimum energy/throughput^2 on the bathtub-shaped E/T^2 vs. VF level curve. Because our threads are generally loosely coupled and we do not scale the speed of the memory domain, we do not require the precise interleaving of *search* and *hold* phases. Instead, each VFI's controller operates independently. Each VFI contains a counter for the number of non-NOP instructions retired, and we assume a mechanism to measure static and dynamic energy consumption as in [9].

For each interval, we compare the E/T^2 value over that interval with that over the previous one. If E/T^2 has decreased, the VFI makes another transition in the same direction as the last one, if possible. If E/T^2 has increased, the VFI makes a transition in the opposite direction of the last one and then holds that level for the next N intervals. Thus, the general behavior is that the controller will perform a search in one direction D_1 until it overshoots the optimal level, then move back to the optimal level and hold. At the end of the hold phase, exploration continues in direction D_2. When the VFI is within a relatively steady execution phase, it will spend $2N$ out of $2N+2$ intervals at the optimal level and two intervals within a single level of the optimal one.

The FSM for *Greedy* is shown in Figure 2; in addition to the states from the *Threshold* controller it has the states `Start Hold on Down` (`SHD`), `Start Hold on Up` (`SHU`), `Hold on Down` (`HD`), and `Hold on Up` (`HU`). Once again, L_n is the VF level used for interval n. Like the other controllers, it spends the first interval at the highest VF level. However, it then unconditionally begins exploring the lower levels. The hold counter H is used to support the *hold* phase.

Parameter	Value
Baseline frequency	4.0 GHz
Technology	45 nm node
Nominal voltages	V_{DD} = 1.0 V, V_{TH} = 0.151 V
DVFS Interval	12.5 μs (50,000 cycles at 4.0 GHz)
L1-I/D caches	64 KB, 64 B blocks, 2-way SA, 2-cycle load-to-use, LRU, 4 R/W
L2 cache	16 MB, 64 B blocks, 16-way SA, 20-cycle hit time, LRU, 16 banks, 1R+1W per bank
Main memory	80 ns for a random access
Pipeline	8 stages deep, 4 instructions wide
ROB/LSQ size	128
Store buffer size	64

Table 1: Processor Parameters

5. VFI CONFIGURATIONS

We consider several different VFI granularities. Finer-grained VFIs are expected to enable better DVFS performance through increased flexibility. The choice of VFI granularity also impacts the number and size of the clock networks in the processor, affecting both clock power and skew. We model the power effect, but conservatively ignore the skew effect, assuming that the baseline VFI processor runs at the same frequency as the reference synchronous processor. Finally, increasing the number of VFIs increases the design complexity.

The L2 and memory interface always form their own clock domain, while the cores are split equally into one, four, or 16 domains. Thus, we have a configuration where every core runs independently, a configuration where all the cores run at the same speed, and an intermediate configuration where the cores are arranged in four clusters of four cores each. For the clustered organization, we consider the case where a cluster is comprised of all the cores with the same *index* mod 4 and the case where cores with consecutive indices are grouped together. Examining both allows us to determine how significant an effect the mapping of threads to cores to VFIs has on DVFS performance.

6. EXPERIMENTAL METHODOLOGY

6.1 Configurations Simulated

We perform our experiments using the processor parameters in Table 1. We simulate an aggressive L2 cache architecture to avoid the shared cache becoming the primary bottleneck in most workloads, which would make it optimal in many scenarios for all cores to run at the lowest VF level.

We simulate several VFI microarchitectures. VFI-B, our VFI baseline, splits the core into 17 clock domains, but runs each at the same frequency as the synchronous baseline. Microarchitectures using DVFS are named VFI-x-y. x specifies the DVFS control algorithm being used ("P" for *PI*, "G" for *Greedy*, and "T" for *Threshold*), while y specifies the number of core VFIs in the design ("1" for a single clock domain containing all cores, "16" for an individual VFI per core, "4M" for four cores per VFI using modulo mapping, and "4C" for four cores per VFI using consecutive mapping).

We experimented to find good values for the DVFS algorithm parameters. For *PI*, we use stability constants K_i = 0.6 and K_p = 0.2. These were found to give better performance than the values of 0.3 and 0.1 suggested by Juang *et al.* [7], probably because our significantly larger threads dis-

Web server (SPECweb99, 16K connections)	
apache	FastCGI, worker threading model
zeus	FastCGI
Online transaction processing (TPC-C, 100 warehouses)	
tpcc_db2	DB2, 64 clients, 450 MB buffer pool
tpcc_oracle	Oracle, 16 clients, 1.4 GB SGA
Decision support systems (TPC-H on DB2)	
tpch_qry1	Scan-dominated, 450 MB buffer pool
tpch_qry2	Join-dominated, 450 MB buffer pool
Scientific	
ocean	1026x1026 grid, 9600 s relaxations, 20K res., err. tol. 1e-07
sparse	4096x4096 matrix

Table 2: Workloads Used

play less short-term variation. We found a reference queue occupancy q_{ref} of 70% to yield good results. For *Greedy*, a hold interval length of five DVFS intervals was used. Finally, the target utilization band for *Threshold* was set to $[0.4, 0.6]$. We initially tried to use a band centered around 70% utilization, but found this to yield high performance degradation for an unacceptably small energy benefit. We hypothesize that the control-theoretic nature of *PI* allows it to use more aggressive targets with less risk of an overshoot in reducing a VFI's VF level.

6.2 Workloads Simulated

We simulate the commercial and scientific workloads in Table 2. Our online transaction processing (OLTP) workloads include TPC-C v3.0 on both *IBM DB2 v8 ESE* and *Oracle 10g Enterprise Database Server*. The Decision Support Systems (DSS) workloads consist of two TPC-H queries, categorized as scan- or join-dominated [10]. *Apache HTTP Server v2.0* and *Zeus Web Server v4.3* are evaluated on SPECweb99 under full saturation by requests. Finally, we simulate two scientific workloads: *ocean* (ocean current simulation) and *sparse* (solves $\mathbf{Ax}=\mathbf{b}$ for sparse $\mathbf{A}$).

Each workload is simulated at several points to provide sampling measurements. Each checkpoint is loaded along with all non-transient state (cache, branch predictor, memory, and disk contents) and detailed simulation is performed for 100 μs (400,000 cycles at 4 GHz). The first 75 μs are used for warmup; statistics are only gathered over the last 25 μs. We use throughput, expressed as the number of non-spin user instructions retired in the last 25 μs, as a proxy for performance. This corresponds roughly to the amount of useful work done for workloads such as TPC-H and our scientific applications, while it has been shown for transaction-oriented workloads such as SPECweb and TPC-C that the number of user instructions per transaction is relatively constant [14] (and thus the user instruction retire rate is proportional to the performance metric of transactions per minute).

7. RESULTS

The VFI configurations are compared on throughput and energy/throughput^2 in Figure 4. The reported values are all normalized with respect to the synchronous baseline with no voltage or frequency scaling. The VFI baseline is shown for reference in each set of results. Because we simulate the same time interval for every configuration, the average power draw is just a constant multiple of the total energy. Paired-measurement sampling [13] was used to generate the results and associated 95% confidence intervals.

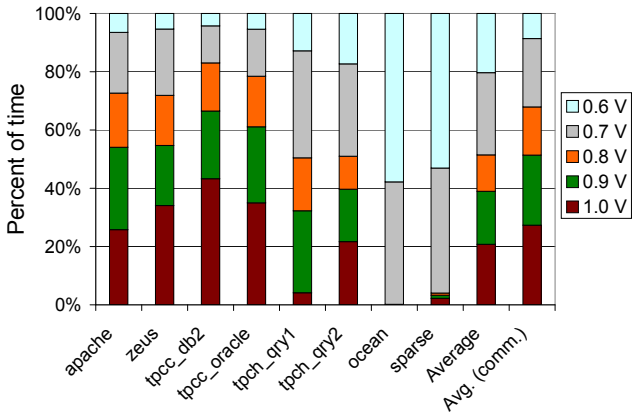


Figure 3: VFI-G-16 utilization of VF levels

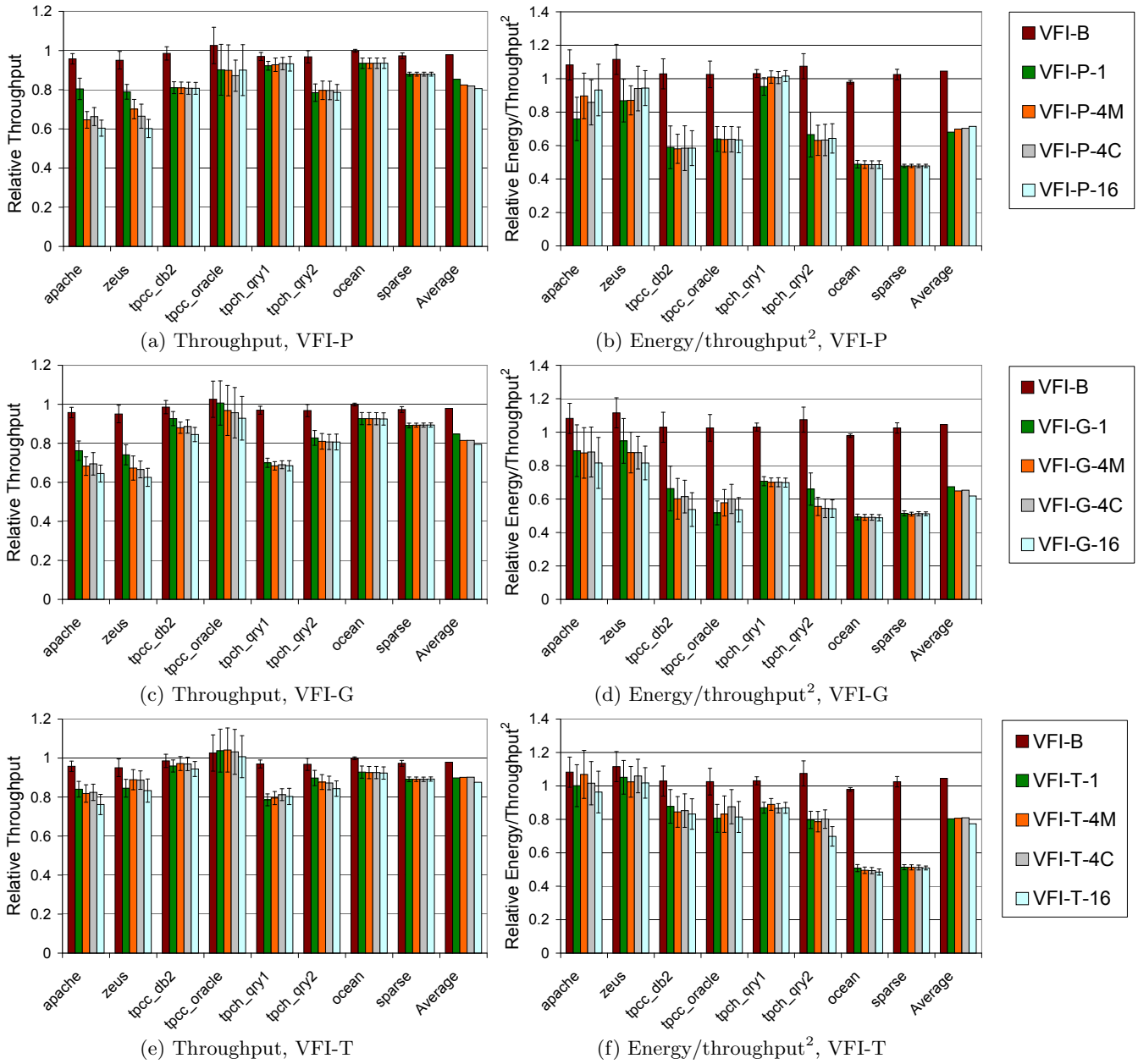
The results for VFI-B show an average performance loss of 2.1% in moving from a fully-synchronous microarchitecture to a VFI one, with the worst case being 5.0% for *zeus*. The reduction in clock power is insufficient to offset the performance loss, resulting in an average E/T^2 increase of 4.5%.

We see that the best scheme overall is VFI-G-16, which divides the processor into independent per-core VFIs and uses the *Greedy* DVFS control algorithm. It is able to achieve an average E/T^2 reduction of 38.2% relative to the synchronous baseline on our mix of commercial and scientific workloads, a significantly larger energy-efficiency gain than that found for small benchmarks and kernels with lightweight threads [7]. One might suspect that this indicates that the original design was highly unbalanced. However, we examined the actual utilization of the levels and saw that this was generally not the case. Figure 3 shows the percentage of time spent at each VF level for each workload under VFI-G-16 along with the overall average and the average when considering only the commercial applications We see that the scientific workloads heavily favor the lower VF levels as a result of their high L2 miss rates (around 10% for both *ocean* and *sparse*, compared with about 5% for the next-worst case, *apache*), but that the commercial applications make good use of all the available levels.

We simulated an oracle that groups all the cores in a single clock domain (and places the L2 and memory controller in a second one) and always chooses the VF level which minimizes E/T^2. Simulating oracles for configurations with more VFIs was intractable due to the number of operating points that would have had to be considered. The oracle we were able to simulate showed an average E/T^2 reduction of 33.6% relative to the synchronous baseline. VFI-G-1, which uses the same VFI configuration, achieves an almost-oracular reduction of 32.6% (although one must consider simulation variability). VFI-G-16 performs better than the coarse-grained oracle, suggesting that there is something to be gained from finer-grained VFIs.

However, we see that the differences between the various VFI configurations are, for the most part, relatively small and not statistically significant at a 95% confidence level. Moreover, simulating more points did not improve the confidence intervals significantly. This suggests that the benefits of per-core DVFS on real workloads are not necessarily as large as may generally be assumed. With this caveat in mind, we will nevertheless examine the trends.

In Figure 4, *Greedy* displays the best E/T^2 for every VFI configuration, highlighting the importance of using an al-



(a) Throughput, VFI-P

(b) Energy/throughput^2, VFI-P

(c) Throughput, VFI-G

(d) Energy/throughput^2, VFI-G

(e) Throughput, VFI-T

(f) Energy/throughput^2, VFI-T

Figure 4: Simulation results, all relative to the synchronous baseline

gorithm that considers the metric of interest directly. This is underscored by the results for *PI*, which show worsening E/T^2 as the VFIs are made more fine-grained. *Greedy*, on the other hand, displays an E/T^2 improvement of 8.3% in moving from the coarse-grained VFI-G-1 to the fine-grained VFI-G-16. The additional flexibility of finer-grained VFIs is wasted on *PI* because the algorithm does not know how to take advantage of it. We do see that *PI* tends to be the most stable of the schemes across VFI configurations in terms of how much the E/T^2 improvement varies, perhaps a benefit of the control-theoretic approach.

As mentioned in Section 6.1, the target utilization for *Threshold* had to be set relatively low, favoring higher frequencies. We see that *Threshold* achieves the lowest performance degradation, but also the lowest E/T^2 improvement.

We considered two different core-to-VFI mappings for the case where the cores are broken into four VFIs. We see in Figure 4 that the effect of the mapping is relatively insignificant on average. The OS does not consider the mapping when it assigns threads to cores, and thus good cases tend to be averaged out with bad ones. We considered a "better" mapping (VFI-G-4B), which chooses the better of the two core-to-VFI mappings on a per-checkpoint basis, and applied it to the best-performing algorithm, *Greedy*. The E/T^2 results are shown in Figure 5, along with those for a corresponding "worse" mapping (VFI-G-4W). Such schemes are clearly unrealizable in hardware, but the OS can approximate the same effect through its mapping of threads to cores. We see that a dynamic hardware/software scheme which could fully emulate VFI-G-4B would reduce E/T^2 by 37.9% relative to the synchronous baseline, resulting in a final E/T^2 only 0.4% worse than that of VFI-G-16, and would offer both reasonable hardware complexity and excellent energy-efficiency. Of course, the degree to which the

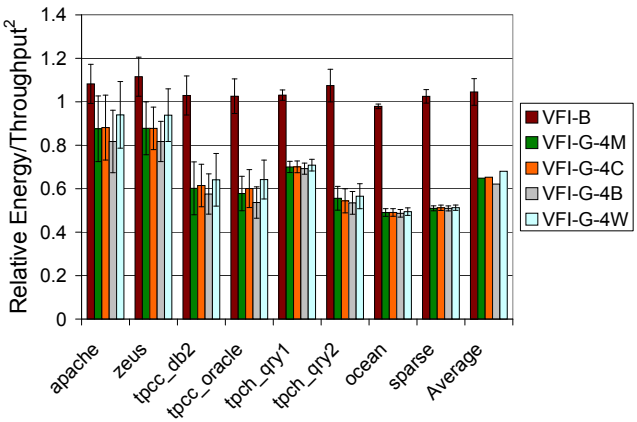


Figure 5: Comparison of core-to-VFI mappings

OS can exploit such effects depends on the characteristics of the variation.

We make some notes regarding the different application classes. There is a great deal of similarity between the two web server applications; this is expected given that serving web pages is a simple task. We also see that workloads from the same application class cluster together in Figure 3. The scientific workloads display the highest utilization of the low two VF levels, followed by the DSS, web server, and OLTP workloads. Moreover, the amount of time spent in the low two levels is very similar for the two workloads in any class. This could be used to tune a design to particular application classes without knowing exactly what software will be run.

8. CONCLUSION

We have made several important new observations regarding fine-grained dynamic voltage/frequency scaling for chip-multiprocessors. We have shown that DVFS can be highly effective in improving the energy-efficiency of CMPs running multithreaded commercial and scientific workloads, with the best scheme we evaluated achieving an energy/throughput^2 reduction of 38.2%. Moreover, we demonstrated that the increasing flexibility offered by moving to finer-grained voltage/frequency islands does not necessarily translate into better energy-efficiency. Even if the DVFS algorithm is able to exploit the extra freedom, the gains are often small and likely not complexity-effective given the number of independent VFIs required. We also showed that a cooperative hardware/software scheme has at least the potential to achieve the energy-efficiency of the most complex hardware-only scheme, but at a substantially lower hardware cost.

9. REFERENCES

[1] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: a scalable architecture based on single-chip multiprocessing. In *ISCA '00: Proceedings of the 27th International Symposium on Computer Architecture*, 2000.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th International Symposium on Computer Architecture*, 2000.

[3] J. A. Butts and G. S. Sohi. A static power model for architects. In *MICRO 33: Proceedings of the 33rd Annual IEEE/ACM international symposium on Microarchitecture*, 2000.

[4] T. Chelcea and S. Nowick. Robust interfaces for mixed systems with application to latency-insensitive protocols. In *DAC '01: Proceedings of the 38th annual Design Automation Conference*, 2001.

[5] J. Dorsey, S. Searles, M. Ciraula, E. Fang, S. Johnson, N. Bujanos, R. Kumar, D. Wu, M. Braganza, and S. Meyers. An integrated quad-core Opteron processor. In *ISSCC '07: IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2007.

[6] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO '06: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006.

[7] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. W. Clark. Coordinated, distributed, formal energy management of chip multiprocessors. In *ISLPED '05: Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, 2005.

[8] J. Li and J. F. Martínez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *HPCA '06: Proceedings of the 12th International Symposium on High-Performance Computer Architecture*, 2006.

[9] G. Magklis, P. Chaparro, J. González, and A. González. Independent front-end and back-end dynamic voltage scaling for a gals microarchitecture. In *ISLPED '06: Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, 2006.

[10] M. Shao, A. Ailamaki, and B. Falsafi. DBmbench: fast and accurate database workload representation on modern microarchitecture. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative Research*, 2005.

[11] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *ISCA '03: Proceedings of the 30th International Symposium on Computer Architecture*, 2003.

[12] E. Talpes and D. Marculescu. Toward a multiple clock/voltage island design style for power-aware processors. *IEEE Trans. Very Large Scale Integr. Syst.*, 13(5), 2005.

[13] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe. Simulation sampling with live-points. In *ISPASS '06: Proceedings of the 2006 International Symposium on Performance Analysis of Systems and Software*, 2006.

[14] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. Simflex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4), 2006.

[15] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural Support for Programming Languages and Operating Systems*, 2004.